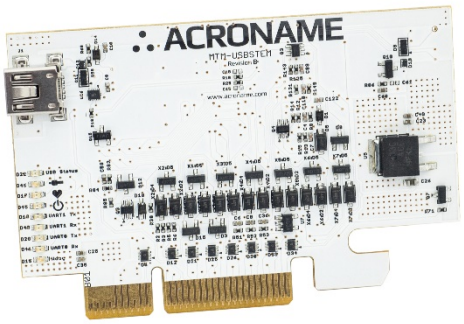


Overview

The MTM-USBStem (S69-MTM-USBSTEM), as part of the Acroname® MTM (Manufacturing Test Module) product series, is a ruggedized general-purpose automation and IO module for use in MTM-based test systems. The MTM-USBStem allows MTM system designers a simple way to add digital and analog IO to their designs.

Ideal for reliability and robustness in manufacturing or R&D environments, the analog and digital outputs on the MTM-USBStem are protected and against overvoltage, short-circuit and overcurrent events. The low profile and small footprint of the MTM-USBStem module makes it ideal for direct integration into test fixtures, thereby eliminating the need for external programmable IO cards or DMMs and their associated cabling.

Built using Acroname's industry-proven and well-adopted BrainStem® technology, resources on the MTM-USBStem are controlled via Acroname's powerful and extensible BrainStem technology and software APIs.

Typical Application

– Functional test-system instrumentation
– In-circuit testing
– Motion control in manufacturing applications
– General test and measurement

System Features

– 15 Digital GPIOs (overvoltage and current protected)
– 3 12-bit ADCs (overvoltage and current protected)
– 1 10-bit DAC (overvoltage and current protected)
– 1 real-time, user-configurable clock
– 1 user-dedicated I²C 400kHz bus
– 1 BrainStem I²C FM+ (1Mbit/s) bus

Description

As part of Acroname's MTM product series, the MTM- USBStem is used to implement general purpose control and automation functions in an MTM-based system. Details on the MTM development platform architecture, BrainStem interface, and APIs are at https://acroname.com/reference.

The MTM-USBStem implements an onboard BrainStem controller running an RTOS (Real-Time Operating System), which provides a host connection, independent operating capability, the BrainStem interface and the MTM resources identified in this datasheet (GPIO, analog IO, I2C, etc.). These resources can be controlled from a host computer over USB or in a network of MTM modules.

MTM-USBStem can be used to make measurements, control signal-conditioning circuitry, or for automation control of mechanical systems via servos or other electromechanical systems. This versatile module is an ideal start and core part of any functional test-system.

IMPORTANT NOTE

The MTM-USBStem utilizes a PCIe connector interface but is for use strictly in MTM-based systems. It should never be installed in a PCI slot of a host computer directly. Insertion into a PC or non-MTM system could cause damage to the PC.

Absolute Maximum Ratings

Stresses beyond those listed under ABSOLUTE MAXIMUM RATINGS can cause permanent damage to the device. These are stress ratings only and functional operation of the device at these or any other conditions beyond those indicated under RECOMMENDED OPERATING CONDITIONS is not implied. Exposure to absolute-maximum rated conditions for extended periods affects device reliability and may permanently damage the device.

Voltage Rating	Minimum	Maximum	Units
Input Voltage, Vsupply	-13.2	13.2	V
I2C0 SDA, SCL	-0.5	13.2	V
UART TX/RX	-0.5	13.2	V
DIO 0-14	-0.5	13.2	V
Module Address 0-3	-0.5	13.2	V
Reset	-0.5	13.2	V
USB D+, D-	-0.5	5.5	V
USB V_{bus}	-0.5	6.0	V
Analog 0-2 – input	-0.5	13.2	V
Analog 3 – output	-0.5	V_{supply}	V

Table 1: Absolute Maximum Ratings

The MTM system is designed to be used in a system where V_{supply} is the highest voltage connected to all MTM modules. Each module is designed to withstand V_{supply} continuously connected to all IOs, excepting those specified above, including accidental reverse polarity connection between V_{supply} and ground (0V). As with all products, care should be taken to properly match interface voltages and ensure a well-architected current-return path to ground. As with all devices utilizing USB interfaces, care should be taken to avoid ground loops within the USB subsystem. When using the USB interface, ground must be at 0V potential to avoid damaging connected host systems.

Handling Ratings

Parameter	Conditions/Notes	Minimum	Typical	Maximum	Units
Storage Temperature, T_{STG}		-10	-	85	°C
Relative Humidity Range	Non-Condensing	5	-	95	%RH
Electrostatic Discharge, V_{ESD}	IEC 61000-4-2, level 4, contact discharge to edge connector interface	-8	-	+8	kV

Table 2: Handling Ratings

Recommended Operating Ratings

Specifications are valid at 25°C unless otherwise noted. Intended for indoor use only.

Parameter	Conditions/Notes	Minimum	Typical	Maximum	Units
Input Voltage, V_{supply}		6.0	-	12.0	V
Voltage to any IO pin		0	-	3.3	V
Voltage to any I2C pin		0	-	3.3	V
Ambient Operating Temperature, T_A	Non-Condensing	0	25	70	°C
Relative Humidity Range	Non-Condensing	5	-	95	%RH

Table 3: Recommended Operating Ratings

Block Diagram

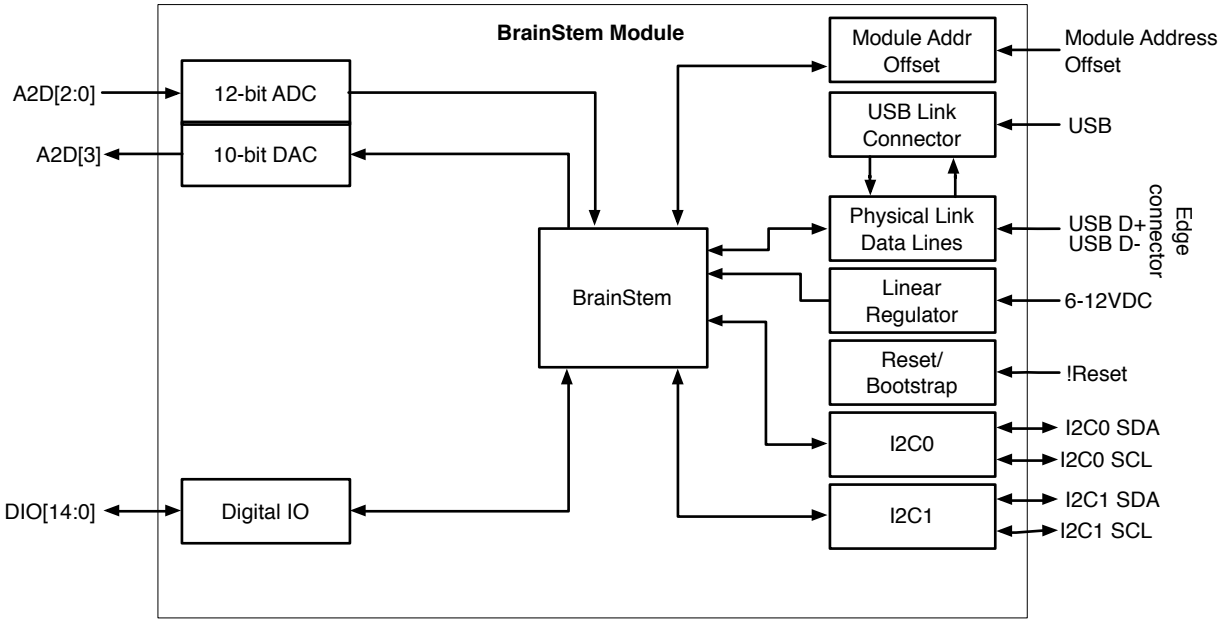


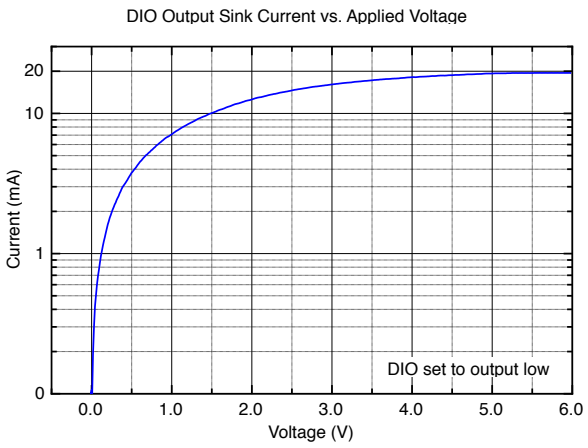
Figure 1: MTM-USBStem Block Diagram

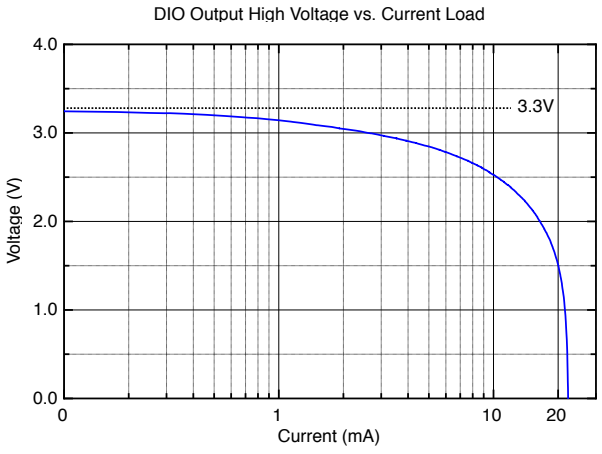
Typical Performance Characteristics

Specifications are valid at 25°C unless otherwise noted. Indoor application use only. Sample rates are typically limited by the USB throughput of the host operating system except where bulk capture is supported.

Parameter	Conditions/Notes	Minimum	Typical	Maximum	Units
Base Current Consumption, I_{supply}	V_{supply} = 6V V_{supply} = 12V	- -	82 83	- -	mA
Reset Low Threshold		-	1.2	-	V
I2C SDA, SCL Pins		0.0	-	3.3	V
Digital Input Logic High, V_{IH}		2.15	-	-	V
Digital Input Logic Low, V_{IL}		-	-	1.1	V
Digital Input Leakage Current	Mode set Input or High-Z	-	110	-	µA
Digital Input Resistance	Mode set Input or High-Z	-	4.25	4.45	MΩ
Digital Output Logic High, V_{OH}		-	3.3	-	V
Digital Output Drive Current	Output high; short to GND V_{output} = 0.9* V_{OH}	- -	20.0 3.15	30.0 -	mA
Digital Output Sink Current	Output low; short to V_{supply}	-	-20.0	-30.0	mA
Digital Output Short Duration	Output high	-	Infinite	-	hours
Digital Output Overvoltage Duration	V_{supply} on pin	-	Infinite	-	hours
Digital Sample Rate[1]	via USB link, C++ Reflex	- -	1000 8200	- -	Hz
Analog Input Voltage		0	-	3.3	V
Analog Input Leakage Current		-	110	-	µA
Analog Voltage Reference Accuracy		-	0.5	-	%
Analog Input Resistance		-	4.25	4.45	MΩ
Analog Input Sample Rate	See Bulk Capture	7.0	-	184.6	kSps
Analog Output Sink Current		-	-	-20.0	mA
Analog Output Source Current	Set at max. output	-	-	19.0	mA
Analog Output Voltage		0.035	-	3.3	V

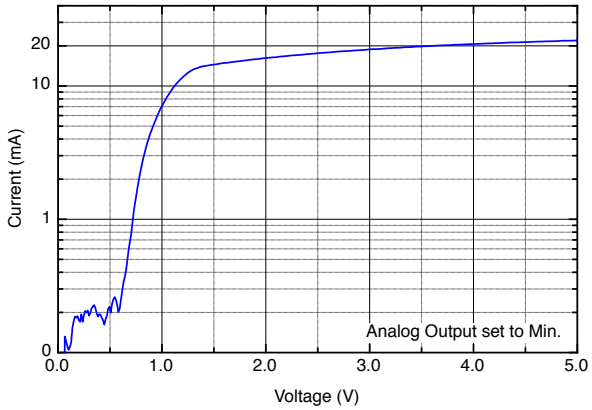
Table 4: Typical Performance Characteristics



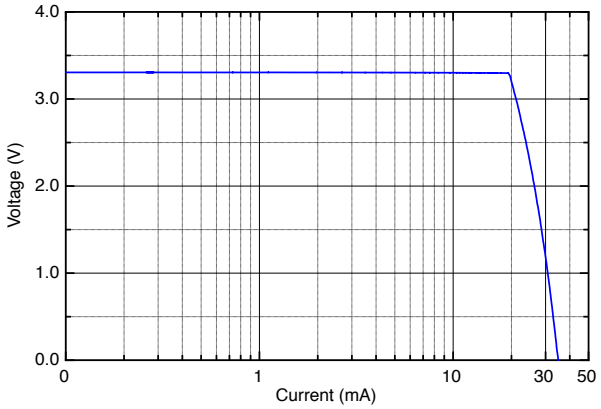


[1] Host dependent, test was done as a single instruction, subsequent instructions may affect performance.

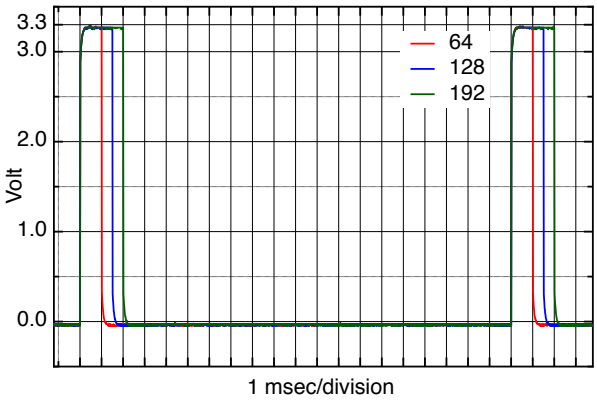
Analog Output Sink Current vs. Applied Voltage



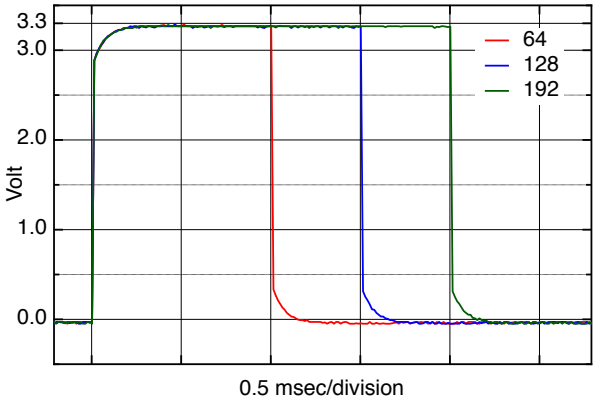
Analog Output (Max.) vs. Current Load



RC Servo Output



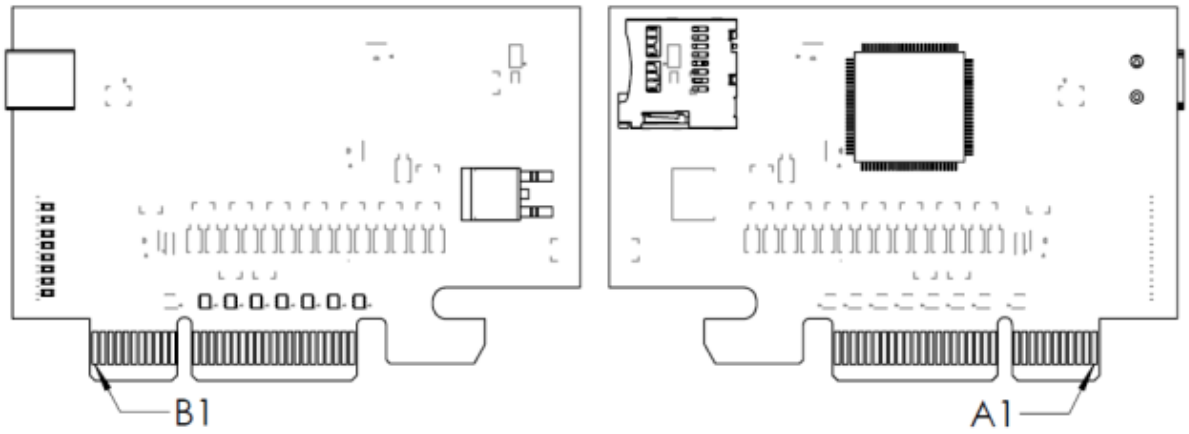
RC Servo Output



Pinout Descriptions

WARNING: MTM modules use a PCIe connector interface that is common in most desktop computers; however, they are NOT intended nor designed to work in these devices. Do NOT insert this product into any PCIe slot that wasn't specifically designed for MTM modules, such as a host PC. Installing this module into a standard PCI slot will result in damage to the module and the PC.

The MTM edge connector pin assignments are shown in the following table. Please refer to Table 3: Recommended Operating Ratings for appropriate signal levels.



Pins Common to all MTM Modules

Side A	Edge Connector Side A Description	Side B	Edge Connector Side B Description
A1	GND	B1	Input Voltage, V_{supply}
A2	GND	B2	Input Voltage, V_{supply}
A3	GND	B3	Input Voltage, V_{supply}
A4	GND	B4	Input Voltage, V_{supply}
A5	Reset	B5	Input Voltage, V_{supply}
A6	GND	B6	Reserved, Do Not Connect
A7	GND	B7	Reserved, Do Not Connect
A8	I2C0 SCL	B8	GND
A9	I2C0 SDA	B9	GND
A10	GND	B10	Reserved, Do Not Connect
A11	GND	B11	Reserved, Do Not Connect
A12	Module Address Offset 0	B12	Module Address Offset 2
A13	Module Address Offset 1	B13	Module Address Offset 3

Table 5: Pins Common to all MTM Modules

Pins Specific to MTM-USBStem

Side A	Edge Connector Side A Description	Side B	Edge Connector Side B Description
A14	Reserved, Do Not Connect	B14	USB Upstream Data + (D+)
A15	Reserved, Do Not Connect	B15	USB Upstream Data - (D-)
A16	Reserved, Do Not Connect	B16	Reserved, Do Not Connect
A17	I2C1 SCL	B17	Reserved, Do Not Connect
A18	I2C1 SDA	B18	Digital IO 0
A19	Reserved, Do Not Connect	B19	Digital IO 1
A20	Reserved, Do Not Connect	B20	Digital IO 2

Side A	Edge Connector Side A Description	Side B	Edge Connector Side B Description
A21	Reserved, Do Not Connect	B21	Digital IO 3
A22	Reserved, Do Not Connect	B22	Digital IO 4
A23	Reserved, Do Not Connect	B23	Digital IO 5
A24	Analog 0	B24	Digital IO 6
A25	Analog 1	B25	Digital IO 7
A26	Reserved, Do Not Connect	B26	Digital IO 8
A27	Reserved, Do Not Connect	B27	Digital IO 9
A28	Reserved, Do Not Connect	B28	Digital IO 10
A29	Reserved, Do Not Connect	B29	Digital IO 11
A30	Reserved, Do Not Connect	B30	Digital IO 12
A31	Analog 2	B31	Digital IO 13
A32	Analog 3	B32	Digital IO 14

Table 6: Pins Specific to MTM-USBStem

Module Hardware and Software Default Values

Software Control

The MTM-USBStem module firmware is built on Acroname's BrainStem technology and utilizes a subset of BrainStem entity implementations that are specific to the hardware's capabilities. Table 7 details the BrainStem API entities and macros used to interface with the MTM-USBStem module. For C and C++ developers, these macros are defined in `aMTMUSBStem.h` from the BrainStem development package. For Python development, the module `MTMUSBStem` class defines the extent of each entity array.

While Table 7 lists the BrainStem API entities available for this module, not all entity methods are supported by the MTM-USBStem. For a complete list of supported entity methods, see Table 12. Note that available method options may vary by entity index, as well as by entity, and calling an unsupported entity option will return an appropriate error (e.g.: `aErrInvalidEntity`, `aErrInvalidOption`, `aErrMode`, or `aErrUnimplemented`) as defined in `aError.h` for C and C++ and the `Result` class in Python.

All API example code snippets that follow are pseudocode loosely based on the C++ method calls - Python and Reflex are similar. Please consult the BrainStem Reference for specific implementation details[3].

Parameter	Index	Macro Name or Implemented Options	Notes
Module Definitions:			
Module Base Address	4	`aMTM_USBSTEM_MODULE_BASE_ADDRESS`	See aMTMUSBStem.h
Entity Class Definitions:			
`digital` Entity Quantity	15	`aMTM_USBSTEM_NUM_DIG`	
`analog` Entity Quantity	4	`aMTM_USBSTEM_NUM_A2D`	
`i2c` Entity Quantity	2	`aMTM_USBSTEM_NUM_I2C`	
`clock` Entity Quantity	1		
`store` Entity Quantity	3	`aMTM_USBSTEM_NUM_STORES`	
`system` Entity Quantity	1		
`timer` Entity Quantity	8	`aMTM_USBSTEM_NUM_TIMERS`	
`app` Entity Quantity	4	`aMTM_USBSTEM_NUM_APPS`	
`pointer` Entity Quantity	4	`aMTM_USBSTEM_NUM_POINTERS`	
`servo` Entity Quantity	8	`aMTM_USBSTEM_NUM_SERVOS`	
`signal` Entity Quantity	5	`aMTM_USBSTEM_NUM_SIGNALS`	

Table 7: MTM-USBStem Hardware and Software Default Values[2]

[2] Refer to `aMTMUSBStem.h` within the BrainStem Development Kit download for actual file.

Capabilities and Interfaces

BrainStem Link and Module Networking

A BrainStem link can be established that will give the user access to the resources available on the MTM-USBStem. The module can then be controlled via a host running BrainStem APIs or operated independently by running locally embedded, user-defined programs based on Acroname's BrainStem Reflex language in the RTOS.

A BrainStem link to the MTM-USBStem can be established via one of three (3) interfaces: the onboard USB connection, the card-edge USB connection, or through another MTM module using the BrainStem protocol (more on this interface below). For the USB connection options, once the MTM-USBStem is attached to a host machine, a user can connect to it via software API:

```
stem.link.discoverAndConnect(linkType,
serialNumber, modelNumber)
```

The MTM-USBStem can also work within a network of other Brainstem modules, such as in a test fixture, to give access to the capabilities of all networked modules. On the MTM platform, networked modules communicate using the Brainstem protocol, which is transmitted over I²C. Each MTM-USBStem is uniquely addressable via hardware or software to avoid communication conflicts on the I²C bus. A software offset can be applied as follows:

```
stem.system.setModuleSoftwareOffset(address)
```

Module Address Hardware Offset Configuration

A hardware offset allows a user to modify the module's address on the BrainStem network. Using hardware offset pins is useful when more than one of the same module type is installed on a single BrainStem network. Applying a different hardware offset to each module of the same type in one network allows for all the modules to seamlessly and automatically configure the network for inter-module communication. Further, modules can be simply swapped in and out of the network without needing to pre-configure a module's address before being added to a network. Finally, when a system has more than one of the same module type in a network, the module address hardware offset can be used to determine the module's physical location and thus its interconnection and intended function. For detailed information on BrainStem networking see the BrainStem Reference[3].

Each hardware offset pin can be left floating or pulled to ground with a 1kΩ resistor or smaller (pin may be directly shorted to ground). Pin states are only read when the module boots, either from a power cycle, hardware reset, or software reset. The hardware offset pin states are treated as a bit state within a 4bit number. This number is multiplied by 2 and added the to the module's base address. The hardware offset calculation is detailed in the following table:

HW Offset Pin				Address Offset	Module Base Address	Final Module Address
3	2	1	0			
NC	NC	NC	NC	0	4	4
NC	NC	NC	1	2	4	6
NC	NC	1	NC	4	4	8
NC	1	NC	NC	8	4	12
1	NC	NC	NC	16	4	20
1	NC	NC	1	(1+8) * 2	4	22

Table 8: Module Address Hardware Offset Examples

Upstream USB Connectivity Options

The MTM-USBStem supports upstream USB connections (to communicate to a host PC) via the mini-B connector, or through pins B14 and B15 of the PCIe edge connector. The module defaults to using the edge connector and will switch to the mini-B connector if 5V is present on V_{bus} at the mini-B connector.

System Entities

Every BrainStem module includes a single System Entity. The System Entity allows access to configuration settings such as the module address and router information, measurements such as input voltage, control over the user LED, and many more.

Saving Entity Settings

Some entities can be configured and saved to non-volatile memory. This allows a user to modify the MTM-USBStem's startup and operational behavior *away* from the factory default settings. Saving system settings creates a new default and often requires a reboot of the MTM-USBStem for changes to take effect; see Table 9: Entity Values Saved by system.save() for relevant settings. Use the following command to save changes to system settings before reboot:

```
stem.system.save()
```

Saved Configurations	
Module Software Offset	I2C Rate
Router Address	Boot Slot
Heartbeat Rate	

Table 9: Entity Values Saved by system.save()

Store Entities

Every BrainStem module includes several Store entities and onboard memory slots to load Reflex files (for details on Reflex, see BrainStem Reference[3]). One Reflex file can be stored per slot. Store[0] refers to the internal flash memory, with 12 available slots, and store[1] refers to RAM, with 1 available slot.

Digital Entities

The MTM-USBStem has fifteen (15) digital input/outputs (DIO) controlled by the digital entity. Each DIO is controllable via software and is independently current limited for both source and sink currents.

All DIO are input and output capable:

```
stem.digital[0].setConfiguration(mode)
stem.digital[0].getConfiguration(mode)
```

The *mode* parameter is an integer that correlates to the following:

Value	Configuration
0	Input
1	Output
2	RC Servo Input
3	RC Servo Output
4	Hi Impedance (Hi-Z)
5	Input with Pull Down
6	Signal Output
7	Signal Input

Table 10: Digital IO Configuration Values

Example: If a digital pin is configured as an output, set the digital logic level:

```
stem.digital[0].setState(level)
```

Example: If a digital pin is configured as an input, read the digital logic level:

```
stem.digital[0].getState(level)
```

If a digital pin is configured in Hi-Z mode its internal circuitry has been disconnected to create a high impedance. There are no functions that can act on this configuration.

Configuring a digital pin as an RCServo input or output requires use of the RCServo Entity. The RCServo input and output modes are only available on a subset of the digital pins. Refer to Table 11: Digital IO Pin Configurations for a complete list. **Note**: Signal and RCServo entities are covered in a subsequent section.

Pin	Input	Output	Hi-Z	RCServo	Signal
DIO 0	Yes	Yes	Yes	Input	-
DIO 1	Yes	Yes	Yes	Input	-
DIO 2	Yes	Yes	Yes	Input	-
DIO 3	Yes	Yes	Yes	Input	-
DIO 4	Yes	Yes	Yes	Output	Input
DIO 5	Yes	Yes	Yes	Output	Input / Output
DIO 6	Yes	Yes	Yes	Output	Input / Output
DIO 7	Yes	Yes	Yes	Output	Input / Output
DIO 8	Yes	Yes	Yes	-	Input / Output
DIO 9	Yes	Yes	Yes	-	-
DIO 10	Yes	Yes	Yes	-	-
DIO 11	Yes	Yes	Yes	-	-
DIO 12	Yes	Yes	Yes	-	-
DIO 13	Yes	Yes	Yes	-	-
DIO 14	Yes	Yes	Yes	-	-

Table 11: Digital IO Pin Configurations

Analog Entities

The MTM-USBStem has three (3) analog inputs (ADC) and one (1) analog output (DAC) all controlled by the analog entity. Each analog is controllable via software and is independently current limited for both source and sink currents.

The analog inputs are connected to a 12-bit ADC, and return a value between 0 and 65535, corresponding to a range of 0-3.3V. The analog inputs can be read back as voltages in microvolts or ADC counts. The analog output is connected to a 10-bit DAC and takes a set value between 0 and 65535, corresponding to a range of 0-3.3V. The analog output may be set in microvolts or DAC counts.

Example: For the analog output (analog[3]), set the DAC value and voltage:

```
stem.analog[0].setValue(counts)
stem.analog[0].setVoltage(microVolts)
```

Example: For the analog inputs (analog[0-2]), read the ADC value:

```
stem.analog[0].getValue(counts)
stem.analog[0].getVoltage(microVolts)
```

The MTM-USBStem's ADC's are also capable of being captured in bulk based on a user defined sample rate. See *Calculating Bulk Capture Sample Rate* for additional information on sample rate settings. Configuring and triggering the bulk capture is accomplished by setting the number of samples and the sample rate, then triggering the capture.

To set the number of samples and the sample rate:

```
s.analog[0].setBulkCaptureNumberOfSamples(1)
s.analog[0].setBulkCaptureSampleRate(7000)
```

where the sample rate is samples per second (Hz). The system can capture any number of samples up the size of the RAM_STORE slot 0 (8191). The capture is then triggered with:

```
stem.analog[0].initiateBulkCapture()
```

Results of the capture are stored in the RAM_STORE slot 0. Results are always stored in ADC counts as two little-endian byte pairs with the second byte the most significant. Computing a sample value from the Store read out is:

```
sampleValue = array[i] + (array[i+1] << 8)
```

Additional information such as requesting capture status and reading back the captured data can be found in the BrainStem Reference[3] and in the BrainStem SDK examples.

I²C Entities

The MTM-USBStem includes access to two separate I²C buses: one operating at a set 1Mbit/s rate, and the other at 400kbits/s.

Note: The 1Mbit/s bus, while user-accessible, is also used for BrainStem network communication so there may be other, non-user-initiated traffic when other BrainStem modules are linked.

Example: Sending 2 bytes (0xABCD) through the I²C bus to a device with address 0x42:

```
stem.i2c.write(0x42, 2, 0xABCD)
```

Example: Reading 2 bytes of data from a device with address 0x42:

```
stem.i2c.read(0x42, 2, buffer)
```

Where *buffer* would be a char array in C++.

The maximum data size for individual read and write operations on an I²C bus through the BrainStem API is 20 bytes. Sending more than 20 bytes of information must be done as an iterated sequence.

Each I²C bus also includes 330Ω pull-up resistors on the SDA and SCL lines, disabled by default. When using the MTM-DAQ-2 in a linked system (communicating over the 1Mbit/s bus), only a single set of pull-ups along the bus should be enabled in order for the I²C bus to work properly (if more than one set is enabled, the lines cannot be pulled low for communication). Similarly, when using a single MTM device to communicate with an external device over the I²C bus, either the internal pull-ups can be enabled, or external hardware pull-ups added.

```
stem.i2c.setPullUp(bEnable)
```

RC Servo Entities

The MTM-USBStem board is equipped with 4 RC servo inputs and 4 RC servo outputs. The RC Servo entity is an overload of the Digital Entity and thus requires proper configuration before this entity can be enabled.

Example: Enabling RC servo input mode for digital pin 0:

```
stem.digital[0].setConfiguration(digitalConfi
gurationRCServoInput)
```

Using the RC servo entity, digital output pins generate pulsed signal based on the RC Servo standard consisting of a period lasting 20ms and high pulse time between 1-2ms. The high time corresponds to a specific position determined by the specific servo being used. RC servo inputs measure this high time and return the corresponding position for a servo.

Example: When operating as an RC servo input, enabling functionality and reading position:

```
stem.RCServo[0].setEnable(bool)
stem.RCServo[0].getPosition(position)
```

Example: When operating as an RC servo output, enabling functionality and setting position:

```
stem.RCServo[4].setEnable(bool)
stem.RCServo[4].setPosition(position)
```

Signal Entities

The MTM-USBStem board has 5 Signal input and 4 Signal outputs. The Signal entity allows generation of square waves by supplying a period and a time high value.

The Signal entity is an overload of the Digital entity and must first be properly configured before it can be enabled. Using Table 11: Digital IO Pin Configurations, configure the correct digital pin as a Signal input:

```
stem.digital[4].setConfiguration(digitalConfi
gurationSignalInput)
```

Now, configure period (t3 time) and the time high value (t2 time):

```
stem.signal[0].setT3Time(period)
stem.signal[0].setT2Time(timeHigh)
stem.signal[0].setEnable(enable)
```

Note: See the BrainStem Reference[3] guide for timing diagram.

Signal and Digital Entities' indices do not align. Counting for the first Signal Entity starts at the first digital pin that is equipped with a Signal overload (Digital 4 = Signal 0, see Table 11: Digital IO Pin Configurations).

Clock Entities

The MTM-USBStem includes a real-time, user-configurable clock entity tracking a time object consisting of year, month, day-of-the-month, hour, minute, and second.

Example: Setting values independently:

```
stem.clock.setYear(year)
stem.clock.setSecond(second)
```

Example: Reading values independently:

```
stem.clock.getYear(year)
stem.clock.getSecond(second)
```

Reflex RTOS

Reflex is Acroname's real-time operating system (RTOS) language which runs in parallel to the module's firmware. Reflex allows users to build custom functionality directly into the device. Reflex code can be created to run autonomously on the module or a host can interact with it through BrainStem's Timer, Pointer App and other entities.

Timer Entities

The Timer entity provides simple scheduling for events in the reflex system. The MTM-USBStem includes 8 timers per reflex. Each timer represents a reflex definition to be executed upon expiration of a running timer. Timers can be controlled from a host, but the reflex code is executed on the device.

Example: Setting up and starting Timer 0 for single use:

```
stem.timer[0].setMode(timeModeSingle)
stem.timer[0].setExpiration(DELAY)
```

Reflex Definition: Timer 0 expiration callback:

```
reflex timer[0].expiration() { //Do Stuff }
```

Pointer Entities

Reflex and the Brainstem module share a piece of memory called the scratchpad which can be accessed via the Pointer Entity. The MTM-USBStem has 4 pointers per reflex which allow access to the pad in a similar manner as a file pointer.

Example: Configure and access the scratchpad in static mode:

```
stem.pointer[0].setMode(pointerModeStatic)
stem.pointer[0].getByte(byte)
```

Reflex Pad: Single unsigned char definition:

```
pad[0:0] unsigned char byteValue
```

App Entities

Apps are reflex definitions that can be directly trigger by the host. These definitions are also capable of passing a parameter into or out of the app reflex definition. The MTM-USBStem is equipped with 4 App Entities per reflex.

Example: Triggering App 0:

```
stem.app[0].execute(parameter)
```

Reflex Definition: App 0 callback:

```
reflex app[0](int appParam) { //Do Stuff }
```

MTM-USBStem Supported Entity Methods Summary

Detailed entity class descriptions can be found in the BrainStem Reference[3]. A summary of MTM-USBStem class options are shown below. Note that when using Entity classes with a single index (i.e., 0), the index parameter can be dropped. For example:

`stem.system[0].setLED(1)` → `stem.system.setLED(1)`

Entity Class	Entity Option	Variable(s) Notes
digital[0-14]	setConfiguration	
	getConfiguration	
	setState	
	getState	
rcservo[0-7]	setEnable	
	getEnable	
	setPosition	Index 4-7 only
	getPosition	
	setReverse	Index 4-7 only
	getReverse	
i2c[0-1]	write	
	read	
analog[0-2]	getValue	
	getVoltage	
	setBulkCaptureSampleRate	
	getBulkCaptureSampleRate	
	setBulkCaptureNumberOfSamples	
	getBulkCaptureNumberOfSamples	
	initiateBulkCapture	
	getBulkCaptureState	
analog[3]	setValue	
	setVoltage	
clock[0]	setYear	
	getYear	
	setMonth	
	getMonth	
	setDay	
	getDay	
	setHour	
	getHour	
	setMinute	
	getMinute	
	setSecond	
	getSecond	
signal[0-5]	setEnable	
	getEnable	
	setInvert	
	getInvert	
	setT3Time	
	getT3Time	
	setT2Time	
	getT2Time	
store[0-2]	getSlotState	
	loadSlot	

Entity Class	Entity Option	Variable(s) Notes
	unloadSlot	
	slotEnable	
	slotDisable	
	slotCapacity	
	slotSize	
system[0]	save	
	reset	
	setLED	
	getLED	
	setBootSlot	
	getBootSlot	
	getInputVoltage	
	getVersion	
	getModuleBaseAddress	
	getModuleSoftwareOffset	
	setModuleSoftwareOffset	
	getModuleHardwareOffset	
	setHBInterval	
	getHBInterval	
	getRouterAddressSetting	
	getModule	
	getSerialNumber	
	setRouter	
	getRouter	
	getModel	
	routeToMe	
timer[0-7]	getExpiration	
	setExpiration	
	getMode	
	setMode	
Pointer[0-3]	getOffset	
	setOffset	
	getMode	
	setMode	
	getTransferStore	
	setTransferStore	
	initiateTransferToStore	
	initiateTransferFromStore	
	getChar	
	setChar	
	getShort	
	setShort	
	getInt	
	setInt	
App[0-3]	execute	

Table 12: Supported MTM-USBStem BrainStem Entity API Methods[3]

[3] See BrainStem software API reference at https://acroname.com/reference for further details about all BrainStem API methods and information.

LED Indicators

The MTM-USBStem board has nine LED indicators to assist with MTM system development, debugging, and monitoring. These LEDs are shown in the diagrams below.

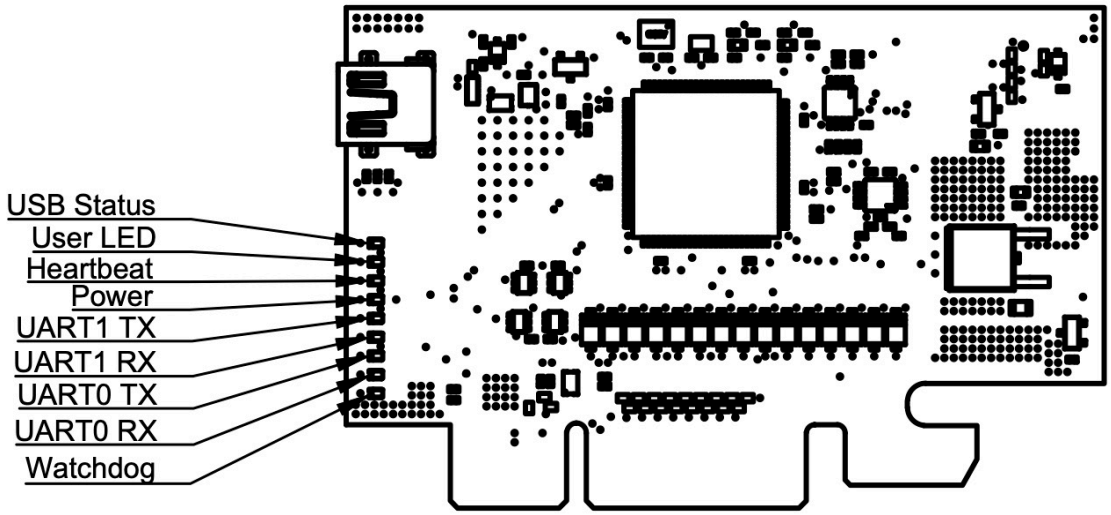


Figure 2: MTM-USBStem LED Indicators

Edge Connector Interface

All MTM products are designed with an edge connector interface that requires a compatible board-edge connector on the carrier PCB. Acroname recommends the through-hole PCI-Express (PCIe) Vertical Connector. The connectors can be combined with an optional retention clip, as shown below. Representative part numbers are shown in Table 13, and equivalent connectors are offered from a multitude of vendors.

Manufacturer	Manufacturer Part Number	Description
Amphenol FCI Samtec	10018784-10201TLF PCIE-064-02-F-D-TH	PCI-Express 64-position vertical connector
Amphenol FCI	10042618-003LF	PCI-Express Retention Clip (optional)

Table 13: PCI-Express Edge Connectors for MTM Products

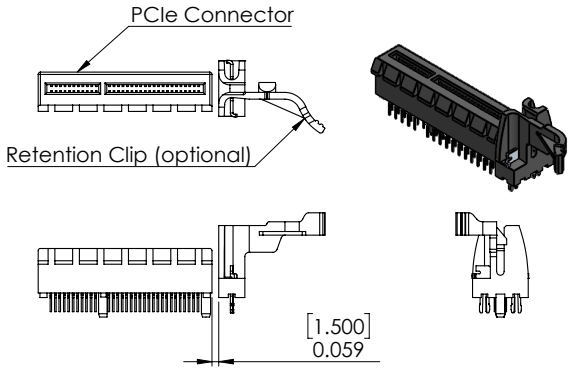


Figure 3: PCIe Vertical Connector with optional Retention Clip

MTM Edge Connector Specifications	Description
Contact Finish	Gold
Card Thickness	0.0625" [1.59mm]
Number of Rows	2
Number of Positions	64
Pitch	0.039" (1.00mm)

Table 14: MTM Edge Connector Specifications

Mechanical

Dimensions are shown in mm. 3D CAD models are available through the MTM-USBStem product page's Downloads section. A 3D CAD viewer with many different CAD model formats available for download is available at https://a360.co/3ddGee5.

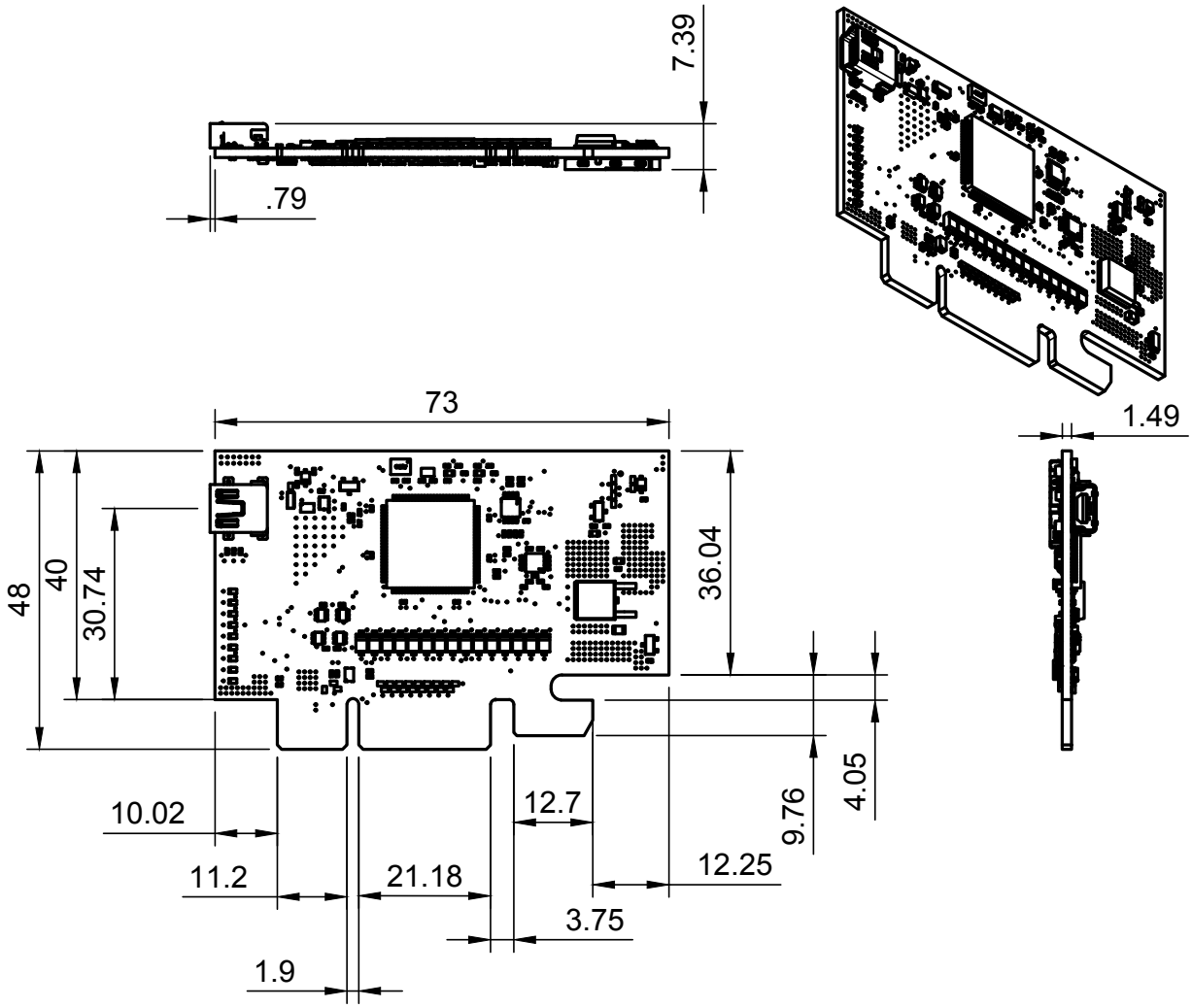


Figure 4: MTM-USBStem Mechanical

Calculating Bulk Capture Sample Rate

Step 1: Calculate Clock Divisor

Cd = Clock Divisor (This value must be rounded up to the nearest whole number

Cf = Clock Frequency = 96,000,000 Hz

n = Number of cycles required for Analog conversion = 65.

Rf = Requested Frequency in Hz

Cd = Cf / (n * Rf)

Step 2: Calculate Actual Bulk Capture Sample Rate

Sr = Sample Rate

Cf = Clock Frequency = 96,000,000 Hz

n = Number of cycles required for Analog conversion = 65.

Cd = Clock Divisor (Calculated in Step 1)

Sr = Cf / (n * Cd)

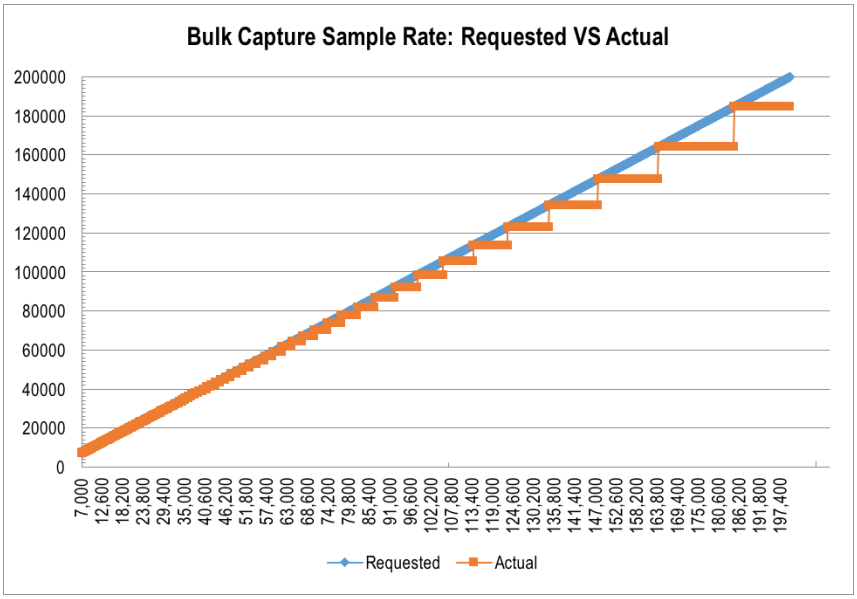


Figure 5: Analog Bulk Capture Actual Sample Rate vs Requested Sample Rate

Document Revision History

All major documentation changes will be marked with a dated revision code

Revision	Date	Engineer	Description
1.0	October 2015	JTD	Initial Revision
1.1	April 2016	JTD	Corrected typographical errors
1.2	September 2016	RMN	Formatting, Error checking, updates
1.3	October 2016	LCD	Updated Overview, Features and Description sections
1.4	October 2016	RMN	Added Bulk Capture information
1.5	December 2016	JG	Clarified I2C pull-ups; update supported API calls
1.6	July 2020	ACRO	Formatting, entity updates, diagram updates

Table 15: Document Revision History